5 CONCLUSIONS AND FUTURE WORK

5.1 Summary of technical contributions

5.2 Summary of empirical findings

5.3 Future Work

Although our experiments are conducted prior to submitting the WebAssembly binary to the FaaS platform, we argue that WebAssembly binary diversification could be implemented at any stage of the FaaS workflow. The same argument holds by using any other diversification tool presented in this dissertation (see Chapter 3).

TODO WASM-MUTATE slicing

We have observed that some transformations can be applied in any order. This means that different sequences of transformations can produce the same binary variant. This often happens when two mutation targets inside the binary are different, such as two disjoint pieces of code. Therefore, a potential parallelization for the baseline algorithm is possible as soon as transformation sequences do not interfere with others.

To further enhance the detection capabilities of MINOS, we believe in binary canonicalization [**?**]. By creating a canonical representation of the malware variant before training and inference, one would help the classifier to better generalize. This is feasible as it is a preprocessing step in the pipeline. We believe this is an interesting direction for future work.

Furthermore, WASM-MUTATE can benefit from the enumerative synthesis techniques employed by CROW and MEWE. Specifically, WASM-MUTATE could incorporate the transformations generated by these tools as rewriting rules.

[0] Compilation probe time 2023/10/23 11:12:04

Moreover, the WebAssembly ecosystem is still in its infancy compared to more mature programming environments. A 2021 study by Hilbig et al. found only 8,000 unique WebAssembly binaries globally[**?**], a fraction of the 1.5 million and 1.7 million packages available in npm and PyPI, respectively. This limited dataset poses challenges for machine learning-based analysis tools, which require extensive data for effective training. The scarcity of WebAssembly programs also exacerbates the problem of software monoculture, increasing the risk of compromised WebAssembly programs being consumed[**?**]. This dissertation aims to mitigate these issues by introducing a comprehensive suite of tools designed to enhance WebAssembly security through Software Diversification and to improve testing rigor within the ecosystem.

Program Normalization WASM-MUTATE was previously employed successfully for the evasion of malware detection, as outlined in [**?**]. The proposed mitigation in the prior study involved code normalization as a means of reducing the spectrum of malware variants. Our current work provides insights into the potential effectiveness of this approach. Specifically, a practically costless process of pre-compiling Wasm binaries could be employed as a preparatory measure for malware classifiers. In other words, a Wasm binary can first be compiled with wasmtime, effectively eliminating approx. 25% of malware variants according to our preservation statistics for wasmtime. This approach could substantially enhance the efficiency and precision of malware detection systems.

Fuzzing WebAssembly compilers with WASM-MUTATE In fuzzing campaigns, generating well-formed inputs is a significant challenge [**?**]. This is particularly true for fuzzing compilers, where the inputs should be executable yet intricate enough programs to probe various compiler components. WASM-MUTATE could address this challenge by generating semantically equivalent variants from an original Wasm binary, enhancing the scope and efficiency of the fuzzing process. A practical example of this occurred in 2021, when this approach led to the discovery of a wasmtime security CVE [**?**]. Through the creation of semantically equivalent variants, the spill/reload component of cranelift was stressed, resulting in the discovery of the before-mentioned CVE.

Mitigating Port Contention with WASM-MUTATE Rokicki et al. [**?**] showed the practicality of a covert side-channel attack using port contention within WebAssembly code in the browser. This attack fundamentally relies on the precise prediction of Wasm instructions that trigger port contention. To combat this security concern, WASM-MUTATE could be conveniently implemented as a browser plugin. WASM-MUTATE has the ability to replace the Wasm instructions used as port contention predictor with other instructions. This would inevitably remove the port contention in the specific port used to conduct the attack, hardening browsers against such malicious maneuvers.